**Projektdokumentation**

Projektarbeit Lernfeld 12

**Thema**

Konzeption, Einrichtung und Umsetzung eines interaktiven Lernspiels zur geografischen Bildung

**Länder-Quiz EU 1.0**

Prüfungsteilnehmer: Abdul Kader Zidan (Projektleiter & Programmierung)

M. Ibrahim Danno (Logik & Programmierung)

Moayad Suleiman (UI-Design)

Laura-Andrea Voicu (Dokumentation &Präsentation)

Praktikumsbetrieb: GFN GmbH

Projektbetreuer: Alfred Walther & Slawomir Banas

Durchführungszeitraum: 16.04.2025 – 09.05.2025

Inhalt

[1 Einleitung 4](#_Toc197494069)

[1.1 Projektumfeld 4](#_Toc197494070)

[1.2 Projektziel 4](#_Toc197494071)

[1.3 Projektbegründung 4](#_Toc197494072)

[1.4 Projektschnittstellen 4](#_Toc197494073)

[1.5 Projektabgrenzungen 4](#_Toc197494074)

[1.6 Allgemeine Bemerkungen 4](#_Toc197494075)

[2 Projektplanung 4](#_Toc197494076)

[2.1 Projektphasen 4](#_Toc197494077)

[2.2 Ressourcenplanung 5](#_Toc197494078)

[2.3 Entwicklungsprozess 5](#_Toc197494079)

[3 Analysephase 5](#_Toc197494080)

[3.1 Ist-Analyse 6](#_Toc197494081)

[3.2 Wirtschaftlichkeit 6](#_Toc197494082)

[3.2.1 Make or Buy-Entscheidung 6](#_Toc197494083)

[3.2.2 Projektkosten 7](#_Toc197494084)

[3.2.3 Amortisationsdauer 7](#_Toc197494085)

[3.3 Fachkonzept 7](#_Toc197494086)

[4 Entwurfsphase 8](#_Toc197494087)

[4.1 Zielplattform 8](#_Toc197494088)

[4.2 Architekturdesign 9](#_Toc197494089)

[4.3 Datenmodell 9](#_Toc197494090)

[4.4 Test-Konzept 10](#_Toc197494091)

[4.5 Pflichtenheft 10](#_Toc197494092)

[5 Implementierungsphase 11](#_Toc197494093)

[5.1 Erstellen der Projektstruktur 11](#_Toc197494094)

[5.2 Implementierung der Datenstruktur 11](#_Toc197494095)

[5.3 Implementierung der Funktionslogik 12](#_Toc197494096)

[5.3.1 Module Entity 12](#_Toc197494097)

[5.3.2 ModuleController 12](#_Toc197494098)

[5.3.3 ModuleRepository 13](#_Toc197494099)

[5.3.4 ModuleLoader 14](#_Toc197494100)

[5.3.5 PageController 14](#_Toc197494101)

[6 Qualitätsmanagement 14](#_Toc197494102)

[7 Dokumentation 15](#_Toc197494103)

[7.1 Entwicklerdokumentation 15](#_Toc197494104)

[7.2 Benutzerdokumentation 16](#_Toc197494105)

[8 Fazit 16](#_Toc197494106)

[8.1 Soll-/Ist-Vergleich 16](#_Toc197494107)

[8.2 Aussichten 17](#_Toc197494108)

[9 Quellenverzeichnis 17](#_Toc197494109)

[10 Anhang 18](#_Toc197494110)

[10.1 Wireframe Startfenster 18](#_Toc197494111)

[10.2 Wireframe Benutzeranleitung 18](#_Toc197494112)

[10.3 Wireframe Quizfenster 18](#_Toc197494113)

[10.4 Wireframe Ergebnisfenster 18](#_Toc197494114)

[10.5 Relational\_Schema 19](#_Toc197494115)

[10.6 country\_from\_flag – Land erkennen anhand der Flagge 19](#_Toc197494116)

[10.7 capital\_from\_country – Hauptstadt zum Land zuordnen 19](#_Toc197494118)

[10.8 Screenshot der Startoberfläche 20](#_Toc197494119)

[10.9 Screenshot der Benutzeranleitung-Oberfläche 20](#_Toc197494120)

[10.10 Screenshot der Quiz-Oberfläche 20](#_Toc197494121)

[10.11 Screenshot der Benutzeranleitung.txt 20](#_Toc197494122)

[10.12 Readme.md 21](#_Toc197494123)

[10.13 Benutzweranleitung.txt 21](#_Toc197494124)

[10.14 Quill Code.py 22](#_Toc197494125)

[10.15 Quill Code.py 23](#_Toc197494126)

[10.16 Quill Code.py 24](#_Toc197494127)

[10.17 Quill Code.py 25](#_Toc197494128)

[10.18 Quill Code.py 26](#_Toc197494129)

[10.19 Test Code.py 27](#_Toc197494130)

[10.20 Ergebnis von Test Code.py 28](#_Toc197494131)

[10.21 Testprotokoll 28](#_Toc197494132)

# Einleitung

* Diese Projektdokumentation beschreibt die Konzeption, Planung, Umsetzung und Evaluation eines selbst entwickelten Softwareprojekts im Rahmen der Ausbildung zum Fachinformatiker für Anwendungsentwicklung.
* Ziel des Projekts ist es, ein geografisches Lernspiel namens „Länder-Quiz“ zu entwickeln, das Wissen über Länder auf interaktive Weise vermittelt.
* Dokumentation folgt den Vorgaben der IHK und bildet den gesamten Projektverlauf vom Kickoff bis zur finalen Präsentation ab .

## Projektumfeld

* Das Projekt "Länder-Quiz" wird im Rahmen der Umschulung zum Fachinformatiker für Anwendungsentwicklung an der GFN GmbH Stuttgart durchgeführt.
* Es simuliert ein reales Softwareprojekt, wie es später im Berufsleben oder im Praktikum im Unternehmen üblich ist.
* Das Projektteam besteht aus vier Mitgliedern und organisiert sich selbstständig mit agilen Methoden (z. B. Kanban über Trello).

## Projektziel

* Ziel des Projekts ist die Entwicklung einer interaktiven Desktop-Anwendung, die geografisches Wissen – wie Hauptstädte, Flaggen – auf spielerische Weise vermittelt.
* Die Anwendung basiert auf Python und verwendet Tkinter zur Umsetzung der Benutzeroberfläche.
* Sie soll lokal lauffähig, benutzerfreundlich und dokumentiert sein.

## Projektbegründung

* Geografisches Wissen wird häufig als trocken empfunden.
* Digitale Lernwerkzeuge sind oft kostenpflichtig, werbeüberladen oder nicht lokal nutzbar.
* Das Projekt soll eine kostenlose, deutschsprachige und lokal installierbare Alternative bieten, die besonders im schulischen Kontext verwendet werden kann.

## Projektschnittstellen

* Das Projekt ist eigenständig und benötigt keine externe Systemanbindung.
* Die Fragen und Antworten werden lokal aus einer JSON-Datei geladen.
* Zur Präsentation werden Office-Programme eingesetzt.

## Projektabgrenzungen

* Kein Online-Modus oder Multiplayer
* Keine komplexen Animationen oder Audioeffekte
* Keine Speicherung oder Analyse der Nutzerdaten

## Allgemeine Bemerkungen

* Das Projekt wurde gemeinsam mit dem Projektbetreuer besprochen und genehmigt.
* Die Arbeit beginnt mit einem Kickoff-Meeting zur Abstimmung der Anforderungen und Rollenverteilung im Team.
* Die gesamte Dokumentation erfolgt gemäß IHK-Richtlinien.
* Das Spiel wurde so gestaltet, dass es offline funktioniert, ohne Internetverbindung. Alle notwendigen Ressourcen (Fragen, Flaggenbilder, Anleitung) sind lokal eingebunden.

# Projektplanung

In diesem Abschnitt zeigen wir, wie wir unser Projekt strukturiert geplant und in klare Phasen unterteilt haben, um effizient und zielorientiert arbeiten zu können.

## Projektphasen

Unser Projekt wurde in mehrere Phasen unterteilt, damit wir strukturiert arbeiten konnten:

1. **Planung**  
   Wir haben die Idee „Länder-Quiz EU“ definiert und ein Ziel festgelegt.  
    siehe **Anhang. 10.6 , 10.7** – JSON-Datenstruktur)
2. **Analyse**  
   Wir haben überlegt, welche Funktionen und Anforderungen wichtig sind.  
    siehe **Anhang. 10.5** – Relationales Schema)
3. **Design**  
   Wir haben Wireframes für die Benutzeroberfläche erstellt.  
    siehe **Anhang 10.1, 10.2, 10.3, 10.4** – Wireframe-Ansichten
4. **Implementierung**  
   Wir haben das Quiz in Python mit Tkinter programmiert.  
    siehe **Anhang 10.14, 10.15, 10.16, 10.17, 10.18** – Python-Code
5. **Testphase**  
   Wir haben das Programm mehrfach getestet.  
    siehe **Anhang 10.19** – TestCode **, 10.20** –TestErgebnis **und 10.21** – Testprotokoll
6. **Dokumentation & Abschluss**  
   Wir haben alle Schritte dokumentiert und die Readme , Benutzeranleitung erstellt.  
    siehe **Anhang 10.12** – Readme, **10.13** – Benutzeranleitung

## Ressourcenplanung

Für das Projekt wurden folgende Ressourcen benötigt:

1. **Teammitglieder**  
   Unser Team besteht aus 4 Personen mit unterschiedlichen Stärken:  
   ➤ Programmierung, Design, Dokumentation, Projektorganisation
2. **Software**
   * **Python 3.10+** mit **Tkinter** für die Programmierung
   * **Editor:** Visual Studio Code
   * **Dateiformate:** .py, .json, .png, .txt, .pdf
3. **Hardware**
   * Windows-PCs mit Internetzugang (nur zur Recherche und Entwicklung)
   * Maus, Tastatur und Monitor
4. **Zeit**
   * Ca. **80 Stunden** für das gesamte Projekt (Planung, Umsetzung, Tests)

## Entwicklungsprozess

Die Entwicklung unseres Länder-Quiz-Programms erfolgte in mehreren klaren und einfachen Schritten:

1. **Ideensammlung und Planung:**  
   In der ersten Phase haben wir gemeinsam im Team überlegt, wie das Spiel aussehen soll. Jeder hat Vorschläge gemacht, welche Funktionen wichtig sind. Danach wurden die Aufgaben unter den Teammitgliedern verteilt.
2. **Gestaltung mit Wireframes:**  
   Um die Benutzeroberfläche besser zu planen, haben wir einfache Skizzen (Wireframes) gezeichnet. So konnten wir besprechen, wo Buttons, Bilder und Texte erscheinen sollen. *(Anhang. 10.1–10.4)*
3. **Programmierung in Python:**  
   Wir haben das Programm mit Python entwickelt. Dafür wurde das Modul **Tkinter** zur Erstellung der Benutzeroberfläche verwendet. Die Fragen wurden aus einer JSON-Datei geladen, die wir manuell erstellt und strukturiert haben. *(Anhang. 10.6 –10.7 und 10.14 –10.18)*
4. **Datenstruktur und Logik:**  
   Die Fragen (insgesamt 80) beinhalten zwei Fragetypen:  
   – Flagge → Land  
   – Land → Hauptstadt  
   Der Zufallsgenerator (random) sorgt für Abwechslung, ohne dass sich Fragen wiederholen.
5. **Testphase:**  
   In mehreren Testdurchläufen wurde das Spiel auf Fehler überprüft. Auch die Darstellung von Bildern, Buttons und Ergebnissen wurde verbessert. Dabei wurde ein **Testprotokoll** verwendet. *(Anhang. 10.19 –10.20 –10.21)*
6. **Benutzerfreundlichkeit:**  
   Wir haben darauf geachtet, dass das Spiel auch für Kinder ab 6 Jahren geeignet ist. Große Buttons, einfache Sprache und klare Rückmeldungen helfen beim Spielen.

*(Anhang 10.8 – 10.9 – 10.10 – 10.11)*

1. **Dokumentation und Anleitung:**  
   Am Ende haben wir eine einfache Benutzeranleitung (Benutzeranleitung.txt + .pdf) sowie ein vollständiges README.md erstellt. *(Anhang 10.12 – 10.13)*  
   So können alle Nutzer das Spiel leicht verstehen und installieren.

# Analysephase

In dieser Phase haben wir den Bedarf für unsere Anwendung analysiert, bestehende Angebote betrachtet und daraus Anforderungen für unser eigenes Produkt abgeleitet.

## Ist-Analyse

* Viele digitale Lernspiele sind heutzutage entweder kostenpflichtig, werbebasiert oder nur online nutzbar. Oftmals sind sie auch ausschließlich in englischer Sprache verfügbar, was den Zugang für viele Nutzerinnen und Nutzer – insbesondere jüngere oder weniger sprachaffine Lernende – erschwert.
* Im schulischen und privaten Bildungsbereich fehlt es häufig an kostenlosen, deutschsprachigen Lernangeboten, die ohne Internetverbindung funktionieren, leicht verständlich und altersübergreifend einsetzbar sind. Gerade Lehrkräfte und Eltern suchen nach solchen Werkzeugen zur gezielten Förderung von Allgemeinwissen.
* Unsere Analyse zeigte, dass es im Bereich der geografischen Bildung keine leicht zugängliche, lokal installierbare Anwendung gibt, die sowohl Kinder ab 6 Jahren als auch Jugendliche und Erwachsene anspricht. Viele bestehende Tools sind zu komplex, überladen oder inhaltlich eingeschränkt.
* Das Projekt „Länder-Quiz EU 1.0“ wurde deshalb bewusst als Offline-Desktop-Anwendung konzipiert, mit einfacher Bedienoberfläche, motivierendem Feedback-System und klar strukturierten Fragen zu europäischen Ländern, Flaggen und Hauptstädten. Es füllt eine erkennbare Lücke im bestehenden Angebot.

## Wirtschaftlichkeit

Die Wirtschaftlichkeit eines Projekts beschreibt, ob sich die Investition von Zeit, Arbeit und ggf. Geld lohnt. Im Rahmen unseres Abschlussprojekts an der GFN GmbH war das Ziel, mit möglichst wenig Mitteln ein möglichst wirkungsvolles Ergebnis zu erzielen – sowohl im Lernnutzen als auch im technischen Aufwand Kostenstruktur:

* Softwarekosten:

Es wurden ausschließlich kostenlose Tools verwendet (Python, Tkinter, Pillow, JSON-Editoren).

* Lizenzen:

Alle verwendeten Inhalte (wie Flaggenbilder) stammen aus freien Quellen oder wurden selbst erstellt.

* Hardware:

Entwicklung erfolgte auf bereits vorhandenen Schulrechnern oder privaten Laptops.

* Personalkosten:

Entfällt, da alle Teammitglieder freiwillig im Rahmen der Ausbildung arbeiteten.

* Zeitaufwand:

Das Projekt war auf etwa 80 Stunden geplant. Dieser Zeitrahmen wurde gut eingehalten.

Durch gute Organisation, Arbeitsteilung und Fokus auf sinnvolle Funktionen wurde effizient gearbeitet.

* Mehrwert/Nutzen:

Das Produkt bietet einen echten Mehrwert für Bildungseinrichtungen, Eltern und Lernende.

Durch die einfache Bedienung und Offline-Funktionalität kann es in vielen Situationen eingesetzt werden.

Auch technisch bietet es Einblicke in GUI-Programmierung, Datenhandling und Projektarbeit.

* Nachhaltigkeit:

Das Spiel ist wartungsarm, da es keine Online-Anbindung oder externe Updates benötigt.

Es kann leicht erweitert werden (z. B. mehr Fragen oder Sprachen).

### Make or Buy-Entscheidung

Vor Beginn eines Projekts muss oft entschieden werden, ob eine Softwarelösung selbst entwickelt („Make“) oder eine bestehende Lösung eingekauft („Buy“) werden soll. Bei unserem Projekt wurde diese Entscheidung bewusst zugunsten der Eigenentwicklung getroffen.

**Gründe für "Make" – also eigene Entwicklung:**

* **Lernziel im Vordergrund**:

Ziel des Projekts war es, den gesamten Softwareentwicklungsprozess zu erlernen – von der Planung über die Umsetzung bis zur Dokumentation. Ein fertiges Produkt zu kaufen würde diesen Lernprozess verhindern.

* **Kostenvorteil**:

Die Eigenentwicklung war komplett kostenfrei. Bestehende Lernsoftwares sind meist kostenpflichtig oder mit Lizenzmodellen verbunden.

* **Individuelle Anpassung:**

Unser Spiel enthält genau die Funktionen, die wir wollten. Eine Fremdsoftware wäre möglicherweise überladen oder unflexibel gewesen.

* **Sprachliche und didaktische Kontrolle:**

Viele vorhandene Tools sind auf Englisch. Durch Eigenentwicklung konnten wir Sprache, Inhalt und Schwierigkeitsgrad gezielt steuern.

* **Offline-Nutzung:**

Die meisten fertigen Tools benötigen Internet. Unser Produkt ist komplett offline nutzbar.

Daher fiel die Entscheidung eindeutig für „Make“, da es die Ziele unseres Ausbildungsprojekts am besten unterstützt.

### Projektkosten

Die Projektkosten orientieren sich an den definierten Projektphasen. Die folgende Tabelle zeigt den geschätzten Zeitaufwand und die zugehörigen Kosten je Phase. Grundlage ist ein Stundensatz von 25 € pro Person. Es wurden keine realen Kosten verursacht – dennoch zeigt die Übersicht den Wert der geleisteten Arbeit.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Projektphase | Beschreibung | Stundenzahl (gesamt) | Satz (€) | Kosten (€) |
| 1. Planung | Ideenfindung, Zieldefinition, Aufgabenverteilung | 10 Stunden | 25 €/h | 250 € |
| 2. Analyse | Anforderungsanalyse, Strukturplanung | 10 Stunden | 25 €/h | 250 € |
| 3. Design | Erstellung von Wireframes und GUI-Konzept | 12 Stunden | 25 €/h | 300 € |
| 4. Implementierung | Programmierung in Python mit Tkinter | 28 Stunden | 25 €/h | 700 € |
| 5. Testphase | Testen des Programms, Bugfixing | 10 Stunden | 25 €/h | 250 € |
| 6. Dokumentation & Abschluss | Projektdokumentation, Benutzeranleitung, Präsentation | 10 Stunden | 25 €/h | 250 € |

**Gesamtkosten: 2.000 €**

Anmerkung:  
- Die Zeiterfassung basiert auf einer Teamgröße von 4 Personen, wobei die Stunden die Gesamtarbeitszeit widerspiegeln.  
- Verwendete Tools (Python, Pillow, Tkinter, VS Code) waren kostenfrei.  
- Keine zusätzlichen Lizenz- oder Hardwarekosten entstanden.  
- Die gesamte Entwicklungsarbeit erfolgte im Rahmen der Umschulung bei der GFN GmbH Stuttgart.

### Amortisationsdauer

Die Amortisationsdauer beschreibt den Zeitraum, in dem sich die Projektkosten durch den erzielten Nutzen ausgleichen. Für das Projekt „Länder-Quiz EU 1.0“ wurde ein Zeitraum von **1,5 Jahren** als realistisches Ziel gesetzt.

**Berechnungsgrundlage:**

* Projektkosten: **2.000 €**
* Jährlicher Nutzen: **ca. 1.350 €**
  + basierend auf der Nutzung durch 15 Schulklassen oder Gruppen pro Jahr
  + angenommene Ersparnis pro Klasse gegenüber kommerziellen Tools: **90 €**

**Rechnung:**

2.000 €1.350 €=1,48 Jahre\frac{2.000\,€}{1.350\,€} = 1{,}48\,Jahre1.350€2.000€​=1,48Jahre

→ Die Amortisation erfolgt nach ca. **1,5 Jahren**, sofern das Spiel regelmäßig eingesetzt wird.

**Ein Bild, das Text, Reihe, Diagramm, parallel enthält.

KI-generierte Inhalte können fehlerhaft sein.**

## Fachkonzept

Das Fachkonzept beschreibt, wie die Anforderungen aus dem Lastenheft aus Sicht der Benutzer und Fachanwender umgesetzt werden sollen. Es verbindet die Wünsche der Nutzer mit einer strukturierten Beschreibung der Funktionen.

1. **Zielgruppe:**
   * Kinder ab 6 Jahren, Jugendliche und Erwachsene
   * Lehrkräfte, Eltern, Schüler
   * Personen, die ihr Wissen über Europa erweitern möchten
2. **Anwendungsziel:**
   * Einfache und motivierende Vermittlung von geografischem Wissen (Flaggen, Hauptstädte)
   * Nutzung im schulischen Kontext, zu Hause oder bei Schulungen
   * Offline-Nutzung ohne Registrierung oder Werbung
3. **Funktionale Anforderungen aus Nutzersicht:**
   * Spielstart über Startfenster mit klarer Auswahl
   * Zwei Fragetypen:
     + Flagge → Land (Benutzer sieht eine Flagge und klickt auf den richtigen Ländernamen)
     + Flagge + Land → Hauptstadt (Benutzer sieht eine Flagge und einen Ländernamen und wählt die passende Hauptstadt)
   * Punktezählung nach jeder Frage
   * Abschlussanzeige mit Auswertung + motivierender Nachricht
   * Möglichkeit zum Neustart oder Beenden
   * Anleitung über Menüpunkt erreichbar
4. **Eingaben und Ausgaben:**
   * **Eingabe:** Klick auf Antwortmöglichkeiten mit Maus
   * **Ausgabe:** Ergebnisanzeige mit Punktezahl, Feedback und nächster Frage
5. **Benutzerführung und Ablauf:**
   * Intuitive Bedienung über große Buttons
   * Klare Navigation: Start 🡪 Quizfenster 🡪 Ergebnisfenster
   * Rückmeldungen in einfacher Sprache
6. **Abnahmekriterien (fachlich):**
   * Das Spiel startet ohne Fehler
   * Die Fragen werden korrekt geladen und angezeigt
   * Antworten werden richtig geprüft
   * Das Ergebnis wird am Ende angezeigt
   * Die Anleitung ist zugänglich und verständlich
   * Das Spiel kann ohne Internet genutzt werden

# Entwurfsphase

In der Entwurfsphase haben wir die technische Zielplattform definiert, die Architektur unserer Anwendung geplant sowie das Datenmodell und das Testkonzept vorbereitet. Diese Grundlagen dienten als Bauplan für die spätere Implementierung.

## Zielplattform

Die Zielplattform für unser Länder-Quiz EU 1.0 ist der Desktop-Computer mit Windows-Betriebssystem. Die Anwendung wurde speziell so entwickelt, dass sie ohne Internetverbindung, ohne Installation und mit möglichst niedrigen Systemanforderungen betrieben werden kann.

**Technische Zielplattform im Detail:**

1. **Betriebssystem:**
   * Windows 10 oder höher (32/64 Bit)
2. **Hardwareanforderungen:**
   * Prozessor: ab Intel i3 oder vergleichbar
   * Arbeitsspeicher: mind. 2 GB RAM
   * Bildschirmauflösung: mind. 1024x760
3. **Softwareanforderungen:**
   * Python 3.x (für Entwickler)
   * Tkinter (in Python enthalten)
   * Pillow (für Bildverarbeitung)
   * Keine Installation notwendig für Benutzer: ausführbare .exe-Datei vorhanden
4. **Besondere Merkmale der Zielplattform:**
   * Offline-fähig: keine Internetverbindung erforderlich
   * Lokale Ausführung ohne Server oder Datenbank
   * Ideal für Schulen, Unterrichtsräume oder private Nutzung
5. **Ausblick für Erweiterungen:**
   * Theoretisch ist eine Portierung auf Linux/MacOS möglich
   * Auch eine Web- oder Mobile-Version wäre langfristig denkbar, war aber nicht Ziel des Projekts

## Architekturdesign

Das Architekturdesign des Projekts "Länder-Quiz EU 1.0" orientiert sich an einer einfachen, modularen Struktur, die eine klare Trennung zwischen Benutzeroberfläche, Datenhaltung und Spiellogik erlaubt. Dadurch wird das Programm übersichtlich, wartbar und erweiterbar.

**1. Modulstruktur des Programms:**

* **Startfenster:**
  + Startbildschirm mit zwei Buttons: „Spiel starten“ und „Benutzeranleitung“
* **Quizfenster:**
  + Anzeige von Fragen, Flaggenbildern und Antwortoptionen
  + Punktesystem, Rückmeldung zur Antwort
* **Ergebnisfenster:**
  + Übersicht der Spielergebnisse, Feedback, Neustart oder Beenden

**2. Dateistruktur und Datenhaltung:**

* **Fragen-Datenbank:** JSON-Datei questions\_with\_ids.json
  + Enthält 80 Fragen (zwei Typen: Flagge → Land, Land → Hauptstadt)
  + Wird beim Start eingelesen und ins Spiel geladen
* **Flaggenbilder:**
  + Gespeichert im lokalen Verzeichnis (PNG-Dateien)
  + Jeder Eintrag in der JSON-Datei enthält einen Bild Pfad

**3. Verwendete Technologien:**

* **Programmiersprache:** Python 3.x
* **GUI-Toolkit:** Tkinter
* **Bildverarbeitung:** Pillow (PIL)
* **Datenstruktur:** JSON (NoSQL-ähnlich, flach)

**4. Interne Logik:**

* Fragen werden zufällig (ohne Wiederholung) geladen
* Nach jeder Antwort erfolgt sofortige Rückmeldung
* Am Ende erfolgt Auswertung und Anzeige

**5. Visualisierung – Programmablauf:**

[Startfenster]

↓

[Quizfenster mit Fragen 1–80] → [Antwortprüfung & Punktestand]

↓

[Ergebnisfenster mit Auswertung & Optionen]

## Datenmodell

Das Datenmodell des Länder-Quiz basiert auf einer einfachen JSON-Struktur, die als flache, nicht-relationale Datenquelle dient. Diese Struktur ist leicht lesbar, gut erweiterbar und ideal für kleine Lernanwendungen geeignet.

**1. Dateiformat:**

* Die Datei questions\_with\_ids.json ist im JSON-Format aufgebaut.
* Jede Frage ist ein eigenständiges Objekt mit folgenden Attributen:

{

"id": 1,

"frage\_typ": "flagge\_land",

"bild": "images/flaggen/deutschland.png",

"frage": "Welche Flagge gehört zu Deutschland?",

"antworten": ["Deutschland", "Frankreich", "Italien", "Spanien"],

"richtige\_antwort": "Deutschland"

}

**2. Feldbeschreibung:**

* id: Eindeutige Identifikationsnummer der Frage
* frage\_typ: Typ der Frage (z. B. flagge\_land oder land\_hauptstadt)
* bild: Pfad zum Bild der Flagge
* frage: Die eigentliche Frage als Text
* antworten: Liste mit 4 Antwortoptionen
* richtige\_antwort: Die korrekte Antwort als Text

**3. Besonderheiten:**

* Die Datenstruktur ist vollständig lokal und benötigt keine Datenbankverbindung
* Erweiterung um neue Fragen ist sehr einfach (Eintrag am Ende der JSON-Liste hinzufügen)
* Die IDs helfen bei der eindeutigen Zuordnung und internen Verarbeitung

**4. Verbindung zur Anwendung:**

* Die Datei wird beim Start des Programms geladen
* Ein Zufallsgenerator (random.shuffle) sorgt für abwechslungsreiche Fragefolgen

Die richtige Antwort wird im Hintergrund geprüft und das Feedback automatisch angezeigt.

## Test-Konzept

Zur Sicherstellung der Qualität und Stabilität der Anwendung wurde ein strukturiertes Testkonzept entwickelt. Es umfasst sowohl technische als auch benutzerorientierte Testkriterien.

**1. Testziel:**

* Überprüfung der grundlegenden Funktionalität (z. B. fehlerfreier Start, Anzeige von Fragen, Ergebnisanzeige)
* Sicherstellung einer intuitiven und verständlichen Benutzerführung
* Identifikation potenzieller Fehlerquellen in der JSON-Datei oder bei der Anzeige

**2. Testmethoden:**

* **Manuelle Tests**: Das Team testete alle Funktionen direkt in der Benutzeroberfläche
* **Blackbox-Verfahren**: Die Anwendung wurde aus Nutzersicht geprüft, ohne den Quellcode zu verändern
* **Fehlertests**: Es wurde auch das Verhalten bei fehlerhaften oder fehlenden Dateien geprüft

**3. Testumfang:**

Basierend auf dem offiziellen Testprotokoll (siehe Anhang), wurden folgende 12 Kriterien getestet:

1. Start der Anwendung ohne Fehler
2. Laden der Fragen aus der JSON-Datei
3. Anzeige der Landesflagge pro Frage
4. Rückmeldung bei Antworten (richtig/falsch)
5. Ergebnisanzeige nach dem Quiz
6. Struktur und Richtigkeit der JSON-Datei
7. Anzeige der beantworteten Fragen
8. Verhalten bei leerer oder fehlerhafter JSON-Datei
9. Startzeit unter 3 Sekunden
10. Fehlermeldung bei fehlender Bilddatei
11. Darstellung der GUI bei verschiedenen Bildschirmgrößen
12. Verständlichkeit der Benutzerführung für Einsteiger

**4. Dokumentation:**

* Alle Testergebnisse wurden in einem separaten Testprotokoll dokumentiert

Status (✅/❌) und Anmerkungen wurden von mehreren Teammitgliedern eingetragen.

## Pflichtenheft

Das Pflichtenheft für das Projekt „Länder-Quiz EU 1.0“ bildet die zentrale Grundlage für die technische Umsetzung. Es basiert auf dem genehmigten Lastenheft vom 17.04.2025 und wurde in enger Abstimmung mit dem Team im Rahmen der Umschulung zum Fachinformatiker erstellt.

**1. Zweck des Pflichtenhefts:**

* Es beschreibt, **wie** die im Lastenheft formulierten Anforderungen technisch umgesetzt werden.
* Dient als verbindliche Grundlage für Planung, Implementierung, Test und Abnahme.

**2. Inhalte des Pflichtenhefts:**

* **Funktionale Anforderungen:**
  + Fragen aus JSON-Datei
  + Anzeige von Flaggen, Antworten, Rückmeldungen
  + Ergebnisanzeige am Ende, Navigation über Mausklicks
* **Nichtfunktionale Anforderungen:**
  + Benutzeroberfläche intuitiv, offline-fähig, werbefrei
  + Technische Anforderungen: Windows 10+, Startzeit < 3s, Größe < 50 MB
  + Datenschutz: keine Internetverbindung, keine Nutzerdaten
* **Technische Rahmenbedingungen:**
  + Verwendung von Python, Tkinter, Pillow, JSON
  + Einsatz von PyInstaller zur Erstellung der .exe-Datei
* **Projektstruktur und Zeitplan:**
  + Erstellt im Team über mehrere Projektphasen von April bis Mai 2025
  + Dokumentation, Wireframes, GUI, JSON-Datenbank und Tests vollständig enthalten
* **Lieferumfang:**
  + EXE-Datei, JSON-Datei, Flaggen-Ordner, Anleitung.pdf, Projektdokumentation
* **Abnahmebedingungen:**
  + Das Spiel startet fehlerfrei, lädt Fragen korrekt, zeigt Flaggen, reagiert richtig auf Antworten und funktioniert offline

**3. Nutzen des Dokuments:**

* Das Pflichtenheft ermöglichte dem Team eine strukturierte Umsetzung des Projekts.
* Es half dabei, Missverständnisse zu vermeiden und die Qualitätsziele konsequent einzuhalten.

# Implementierungsphase

In dieser Phase wurde die technische Umsetzung unserer Anwendung durchgeführt. Wir begannen mit der Strukturierung des Projektverzeichnisses, gefolgt von der Entwicklung der Datenstruktur und der Funktionslogik. Der Fokus lag dabei auf Übersichtlichkeit, Wiederverwendbarkeit und Stabilität des Codes.

## Erstellen der Projektstruktur

Zu Beginn der Implementierungsphase wurde eine strukturierte Projektordnerstruktur erstellt, um den Entwicklungsprozess effizient und übersichtlich zu gestalten. Die Ordner und Dateien spiegeln dabei den vollständigen Aufbau und Ablauf der Anwendung wider.

**1. Projektverzeichnisübersicht:**

Länder\_Quiz\_Program/

├── .venv/ ← Virtuelle Python-Umgebung

├── build/ ← Build-Dateien erstellt durch PyInstaller

├── dist/ ← Kompilierte Anwendung (main.exe)

├── flage/ ← Flaggenbilder (PNG)

├── Benutzeranleitung.pdf ← Anleitung im PDF-Format

├── Benutzeranleitung.txt ← Anleitung als Textdatei

├── main.py ← Hauptprogramm (Startpunkt der App)

├── main.spec ← PyInstaller-Konfigurationsdatei

├── questions\_with\_ids.json ← JSON-Datei mit allen Quizfragen

├── README.md ← Übersicht und Hinweise zum Projekt

**2. Erklärung der wichtigsten Komponenten:**

1. **.venv/**: Enthält alle Bibliotheken, die im virtuellen Python-Umfeld installiert

wurden (z. B. tkinter, pillow)

1. **build/**: Temporäre Dateien, die beim Erstellen der .exe erzeugt wurden
2. **dist/**: Enthält die fertige ausführbare Datei (main.exe) für Windows
3. **flage/**: Beinhaltet über 40 Flaggenbilder im PNG-Format, benannt nach
4. dem jeweiligen Land
5. **main.py**: Das zentrale Python-Skript, welches die grafische Oberfläche startet
6. und das Spiel steuert
7. **questions\_with\_ids.json**: Strukturierte Datenbasis für das Quiz mit Fragen, Antworten und Bildpfaden
8. **README.md**: Enthält Hinweise zur Nutzung und zum Aufbau des Projekts

**3. Ziel der Struktur:**

* Klare Trennung zwischen Code, Bildern, Daten, Ausgabedateien und Dokumentation
* Unterstützung des Build-Prozesses mit PyInstaller
* Einfache Wartung und Erweiterung des Projekts in Zukunft

## Implementierung der Datenstruktur

Die Datenstruktur des Länder-Quiz basiert vollständig auf einer gut strukturierten JSON-Datei namens questions\_with\_ids.json. Diese Datei enthält insgesamt **80 Einträge**, aufgeteilt in **zwei verschiedene Entitätstypen (Fragetypen)**:

**1. Zwei Entitätstypen (Fragearten):**

* [**a) country\_from\_flag – Lan****d erkennen anhand der Flagge**](#country_from_flag_Land_erkennen)
* [**b) capital\_from\_country – Hauptstadt zum Land zuordnen**](#capital_from_country_Hauptstadt)

**2. Feldbeschreibung beider Typen:**

* type: Art der Frage (country\_from\_flag oder capital\_from\_country)
* question: Der Fragetext
* image: Pfad zur Flaggenbild-Datei (PNG) im flage/-Ordner
* choices: Vier Antwortoptionen (Multiple Choice)
* answer: Richtige Antwort als Text
* id: Eindeutige Nummer zur Identifikation der Frage

**3. Datenverteilung:**

* **40 Einträge mit type country\_from\_flag**
* **40 Einträge mit type capital\_from\_country**
* Gesamt: 80 Fragen

**4. Verarbeitung im Code:**

* Die Datei wird beim Programmstart geladen:

with open("questions\_with\_ids.json", "r", encoding="utf-8") as file:

all\_questions = json.load(file)

* Der jeweilige Typ wird geprüft, um die passende Darstellung zu wählen:

if question["type"] == "country\_from\_flag":

# Anzeige der Flagge und Ländernamen

elif question["type"] == "capital\_from\_country":

# Anzeige von Land und Hauptstadt-Auswahl

* Die Bilder werden über Pillow eingebunden:

img = Image.open(question["image"])

**5. Vorteile:**

* Flexible Erweiterung möglich
* Einheitliche Struktur für beide Fragetypen
* Klare Trennung zwischen Inhalt (JSON) und Logik (Python)

## Implementierung der Funktionslogik

### Module Entity

Das Modul **Entity** dient zur einheitlichen Modellierung der Datenstruktur einer Quizfrage. Es kapselt die Informationen einer einzelnen Frage in einer **Klasse**, um den Zugriff und die Weiterverarbeitung zu vereinfachen und zu standardisieren.

Im aktuellen Projekt haben wir **zwei Fragetypen (Entities)**, wie sie in der JSON-Datei definiert sind:

1. **country\_from\_flag** → Frage: „Wie heißt dieses Land?“ mit einer Flagge als Bild
2. **capital\_from\_country** → Frage: „Was ist die Hauptstadt von …?“ mit einem Ländernamen und Flagge

Zur einheitlichen Verwaltung dieser Daten wurde ein **Frage-Objektmodell** in Form der Klasse QuestionEntity definiert (implizit durch Verwendung in main.py).

**Beispiel einer möglichen Klasse (Pseudocode):**

class QuestionEntity:

def \_\_init\_\_(self, id, question, choices, answer, image, qtype):

self.id = id

self.question = question

self.choices = choices

self.answer = answer

self.image = image

self.type = qtype

Diese Klasse fasst alle Eigenschaften einer Frage in einem Objekt zusammen. Sie ermöglicht:

* einheitlichen Zugriff auf Datenfelder
* saubere Übergabe von Fragen zwischen Modulen
* spätere Erweiterbarkeit bei neuen Fragetypen

**Integration im Projekt:**

Obwohl im main.py die JSON-Daten direkt als Dictionary genutzt werden, könnten zukünftige Erweiterungen wie z. B. das Filtern nach Fragetyp oder Sprache durch eine **Klassenstruktur** stark vereinfacht werden.

Das **relational-logische Schema** (siehe Kapitel 4.3 und Anhang) bestätigt, dass jede Frage ein vollständiges Dokument mit allen relevanten Feldern darstellt – daher ist das Entity-Modul ein zentrales Element für die Datenrepräsentation.

### ModuleController

Das **Controller-Modul** übernimmt die Steuerung des Spielflusses im Länder-Quiz und bildet damit das Herzstück der Anwendungslogik. Es sorgt für die Verknüpfung zwischen Benutzeroberfläche (GUI) und den Daten (Fragen + Bilder), sowie für die Navigation durch das Quiz.

Im vorliegenden Code (main.py) ist die Controller-Logik in zwei Klassen integriert:

1. **StartWindow** → Startseite mit Begrüßung und Hilfe
2. **QuizWindow** → Steuerung der Quizfragen und Spielinteraktion

* **Wichtige Funktionen des Controllers:**
* **StartWindow**
* Initialisiert das Startfenster mit Titel, Beschreibung, Start-Button und Anleitung
* Startet das Quiz mit dem Aufruf QuizWindow() bei Klick auf "Quiz starten"
* Öffnet Benutzeranleitung (PDF oder TXT)
* **QuizWindow**
* **Lädt neue Fragen:**

q = random.choice(all\_questions)

* **Verwaltet den Spielzustand:**
  + - aktuelle Frage (self.current\_question)
    - Richtige und falsche Antworten (score\_right, score\_wrong)
    - Liste asked\_ids für bereits gestellte Fragen
* **Steuert die Anzeige:**
  + - Frage und Antwortoptionen
    - Bildanzeige über ImageTk
    - Aktualisierung des Punktestands
* **Überprüft die Antwort:**

if selected == correct:

self.score\_right += 1

else:

self.score\_wrong += 1

* **Zeigt das Ergebnis:**
* Weiterleitung an die Klasse ResultWindow
* **Spielfluss-Steuerung im Überblick:**

StartWindow → QuizWindow → Ergebnisanzeige

| | ↑

+--- Hilfe ---+ |

(PDF/TXT) Quiz erneut starten

* **Vorteile dieser Controller-Logik:**
* Einfache Navigation und Interaktion
* Zentralisierte Steuerung aller Spielfunktionen
* Klare Trennung der GUI-Fenster mit jeweiliger Funktionalität

### ModuleRepository

Das **Repository-Modul** übernimmt die Verantwortung für das Laden, Verwalten und Bereitstellen der Quizdaten, welche in einer JSON-Datei gespeichert sind. Es dient als Schnittstelle zwischen der Datenquelle und dem Controller, um die Daten strukturiert und sauber verfügbar zu machen.

In unserem Projekt wird die Repository-Funktionalität direkt im Hauptskript main.py implementiert.

**Ladevorgang der Daten:**

with open("questions\_with\_ids.json", "r", encoding="utf-8") as file:

all\_questions = json.load(file)

* Der gesamte Inhalt der JSON-Datei wird in eine Liste all\_questions geladen.
* Diese Liste enthält **80 Quizfragen**, die zwei Typen umfassen:
  + - country\_from\_flag (Flagge → Land)
    - capital\_from\_country (Land → Hauptstadt)

**Datenstruktur der Fragen:**

Jede Frage besteht aus den folgenden Feldern:

* type: Art der Frage (z. B. country\_from\_flag)
* question: Formulierte Frage
* image: Pfad zum Bild der Flagge
* choices: Liste der möglichen Antworten
* answer: Richtige Antwort
* id: Eindeutige ID der Frage

Diese strukturierte Form ermöglicht dem Controller einen einfachen Zugriff auf alle Informationen zur Darstellung im Quiz.

**Vorteile des Repository-Moduls:**

* Zentrale Datenquelle
* Leicht erweiterbar (z. B. neue Fragetypen, zusätzliche Informationen)
* Klarer Trennpunkt zwischen Datenhaltung und Spielsteuerung

**Erweiterungsmöglichkeit:**

Zukünftig könnte das Repository in ein eigenes Modul ausgelagert werden (z. B. repository.py), um:

* Datenvalidierung einzuführen
* Zufällige Auswahl als Methode bereitzustellen (get\_random\_question())
* Filterfunktionen für bestimmte Fragetypen zu integrieren

### ModuleLoader

Das **Loader-Modul** ist für das Laden und die Aufbereitung visueller Ressourcen zuständig, insbesondere der Flaggenbilder, die bei jeder Quizfrage angezeigt werden. Auch wenn es in unserem Projekt kein separates Modul loader.py gibt, ist die Funktionalität klar in main.py implementiert.

**Bildlade-Logik mit Pillow:**

Die Bibliothek Pillow (Import from PIL import Image, ImageTk) wird verwendet, um die PNG-Bilder aus dem Verzeichnis flage/ zu laden und für die Anzeige in der Benutzeroberfläche aufzubereiten.

**Beispielcode:**

img = Image.open(image\_path) # Bild laden

img = img.resize((300, 200)) # Bildgröße anpassen

img = ImageTk.PhotoImage(img) # Umwandlung für tkinter

self.flag\_image\_label.configure(image=img)

self.flag\_image\_label.image = img # Referenz speichern

**Ladefunktionen im Ablauf:**

* Die Variable image\_path wird aus dem JSON-Datensatz gelesen
* Das Bild wird geöffnet, auf feste Maße skaliert (300x200px)
* Danach wird es in der GUI (Label) angezeigt

**Fehlerbehandlung:**

* Falls das Bild fehlt oder defekt ist, zeigt das Programm eine Meldung (try/except empfohlen für Erweiterung)

**Vorteile und mögliche Erweiterungen:**

* Trennung der Bildverarbeitung vom Fragemodul
* Möglichkeit zur Wiederverwendung in anderen GUI-Fenstern

### PageController

Der **PageController** koordiniert den Wechsel zwischen den verschiedenen GUI-Seiten des Programms: **Startseite**, **Quizseite** und **Ergebnisanzeige**. Obwohl es im Projekt keinen expliziten page\_controller.py gibt, ist die Logik dieser Komponente in den Klassen StartWindow, QuizWindow und ResultWindow direkt integriert.

**Ablauf der Navigation:**

* StartWindow → stellt die Startseite dar und enthält den Button zum Starten des Quiz
* Bei Klick wird QuizWindow instanziiert und das Fenster gewechselt
* QuizWindow lädt und zeigt nacheinander Fragen und Bilder an
* Nach Beantwortung aller Fragen wechselt das Programm zu ResultWindow

**Steuerlogik im Code:**

# Von Start zu Quiz:

self.root.destroy()

QuizWindow()

# Von Quiz zu Ergebnis:

self.window.destroy()

ResultWindow(score\_right=self.score\_right, score\_wrong=self.score\_wrong)

**Hauptaufgaben des (impliziten) PageControllers:**

* Zerstörung des alten Fensters zur Speicheroptimierung
* Start des neuen Fensters mit ggf. Parameterübergabe (z. B. Punktestand)
* Einfache lineare Navigation ohne Zurückspringen

**Vorteile dieses Ansatzes:**

* Klare Trennung der Seiten und Zuständigkeiten
* Verständlicher Ablauf für Benutzer und Entwickler
* Einfache Erweiterbarkeit (z. B. Anleitung, Hilfe, Wiederholen)

# Qualitätsmanagement

Das Qualitätsmanagement war ein zentraler Bestandteil während der Entwicklung des Projekts „Länder-Quiz EU 1.0“. Ziel war es, eine fehlerfreie, benutzerfreundliche und altersgerechte Anwendung zu erstellen, die den Anforderungen aus dem Pflichtenheft entspricht.

**Qualitätssicherungsmaßnahmen:**

1. **Anforderungsüberprüfung:**
   * Alle funktionalen und nicht-funktionalen Anforderungen aus dem Pflichtenheft wurden regelmäßig kontrolliert.
   * Insbesondere die Startzeit, Speichergröße und Offlinefähigkeit wurden in der finalen Version erfüllt.
2. **Testphase mit Testprotokoll:**
   * Es wurde ein detailliertes Testprotokoll erstellt (siehe Anhang), in dem über **12 Prüfpunkte** dokumentiert wurden.
   * Die Tests wurden auf verschiedenen Windows-Geräten durchgeführt.
   * Überprüfte Punkte: Ladezeiten, Bildanzeige, Buttonfunktionen, Ergebnisausgabe, Fehlerfreiheit
3. **Fehlererkennung und -behebung:**
   * Kleinere Fehler wurden während der Entwicklung mehrfach korrigiert (z. B. bei der Anzeige von Umlauten oder Sonderzeichen in Ländernamen)
   * Die Zufallsauswahl wurde optimiert, um Wiederholungen zu vermeiden
4. **Benutzerfreundlichkeit:**
   * Das Spiel wurde auf Kinderfreundlichkeit geprüft (ab 6 Jahre):

* Große Buttons, einfache Sprache, klare visuelle Rückmeldungen

1. **Teaminterne Qualitätssicherung:**
   * Jedes Teammitglied prüfte die jeweilige Komponente (z. B. GUI, Datenbank, Navigation)

* Die Dokumentation wurde gemeinsam korrigiert und ergänzt

**Ergebnis:**

* Alle Anforderungen wurden erfolgreich umgesetzt
* Das Spiel läuft stabil, fehlerfrei und ohne externe Abhängigkeiten
* Die Qualität wurde durch strukturiertes Testen und gezielte Optimierung sichergestellt

# Dokumentation

In diesem Abschnitt stellen wir die technische Dokumentation für Entwickler sowie die Nutzerdokumentation für Endanwender vor. Ziel ist es, sowohl die interne Logik als auch die Bedienung des Programms transparent und verständlich darzustellen.

## Entwicklerdokumentation

Die Entwicklerdokumentation beschreibt die interne Struktur und Funktionsweise des Länder-Quiz EU 1.0. Sie richtet sich an Programmierer, die den Quellcode verstehen, erweitern oder warten möchten.

**Projektübersicht:**

* **Programmiersprache:** Python 3
* **GUI-Toolkit:** Tkinter (Standardbibliothek)
* **Bildverarbeitung:** Pillow (PIL)
* **Datenformat:** JSON für Quizfragen
* **Kompatibilität:** Windows (.exe erzeugt mit PyInstaller)

**Hauptdateien:**

|  |  |
| --- | --- |
| **Datei** | **Beschreibung** |
| main.py | Hauptlogik, GUI-Steuerung, Spielablauf |
| questions\_with\_ids.json | Datenbank mit 80 Quizfragen (2 Typen) |
| Benutzeranleitung.pdf.txt | Anleitung für Benutzer |
| README.md | Überblick über Projektstruktur und Nutzung |
| flage/ | Bilder der Flaggen (PNG) |
| test-main.py | Testcode zur Überprüfung der Programmfunktionen |

**Funktionslogik im Überblick:**

1. **StartWindow:**
   * Begrüßung, Anleitung anzeigen, Quiz starten
2. **QuizWindow:**
   * Frage + Bild laden, Antworten anzeigen, Bewertung, Punktestand aktualisieren
3. **ResultWindow:**
   * Anzeige der erreichten Punkte nach Quizende

**Datenfluss:**

StartWindow → QuizWindow → ResultWindow

| | ↑

Anleitung Bewertung Restart möglich

**Datenstruktur im Detail:**

* JSON-Datei enthält strukturierte Fragen mit Feldern:
  + type, question, choices, answer, image, id
* Zwei Fragetypen:
  + country\_from\_flag
  + capital\_from\_country

**Wichtige Funktionen im Code:**

with open("questions\_with\_ids.json", "r", encoding="utf-8") as file:

all\_questions = json.load(file)

img = Image.open(image\_path)

img = img.resize((300, 200))

img = ImageTk.PhotoImage(img)

**Build-Prozess:**

* Verwendung von PyInstaller zur Erstellung der .exe-Datei:

pyinstaller --onefile --noconsole main.py

**Bibliotheken:**

* tkinter: GUI
* PIL (Pillow): Bildbearbeitung
* random: Zufallsauswahl
* json: Laden der Fragen
* os: Öffnen von Anleitungen

## Benutzerdokumentation

Die Benutzerdokumentation richtet sich an Endanwender, die das Länder-Quiz EU 1.0 verwenden möchten. Ziel ist es, den Einstieg und die Nutzung so einfach wie möglich zu gestalten. Die Dokumentation liegt sowohl als **Textdatei (Benutzeranleitung.txt)** als auch als **PDF-Datei** vor.

**Installation:**

1. Der Folder Länder-Quiz EU1.0 muss lokal auf dem Rechner gespeichert werden
2. Ein Doppelklick auf main.exe startet das Programm
3. Es ist keine zusätzliche Software oder Internetverbindung erforderlich

**Programmnavigation:**

**1. Startfenster (***Anhang 10.8 – 10.9***)**

* Begrüßungstext
* Schaltfläche „Quiz starten“ zum Beginnen des Spiels
* Schaltfläche „Hilfe“ zum Öffnen der Anleitung (PDF/TXT)

**2. Quizfenster (***Anhang 10.10***)**

* Anzeige einer Frage mit vier Antwortmöglichkeiten
* Anzeige der Flagge oder des Landes
* Nach Auswahl einer Antwort erfolgt sofort Feedback
* Der Punktestand wird mitgezählt

**3. Ergebnisfenster (***Anhang 10.11***)**

* Anzeige der Gesamtpunktzahl
* Differenzierung zwischen richtigen und falschen Antworten
* Möglichkeit zum Neustart oder Schließen

**Tipps zur Benutzung:**

* Das Spiel eignet sich für Kinder ab 6 Jahren
* Die Sprache ist bewusst einfach gehalten
* Flaggenbilder befinden sich im Verzeichnis flage/

**Enthaltene Dateien für Nutzer:**

|  |  |
| --- | --- |
| **Datei** | **Beschreibung** |
| main.exe | Startbare Anwendung |
| Benutzeranleitung.txt | Anleitung als Text |
| Benutzeranleitung.pdf | Anleitung mit Grafiken |
| flage/ | Bilderverzeichnis |

# Fazit

Zum Abschluss des Projekts reflektieren wir die Umsetzung im Vergleich zu den geplanten Anforderungen und geben einen Ausblick auf mögliche Weiterentwicklungen.

## Soll-/Ist-Vergleich

Der Soll-/Ist-Vergleich zeigt, inwieweit die im Lastenheft und Pflichtenheft definierten Anforderungen umgesetzt wurden.

| **Kriterium** | **Soll-Zustand (geplant)** | **Ist-Zustand (realisiert)** |
| --- | --- | --- |
| Spielart | Zwei Fragetypen  (Flagge → Land, Land → Hauptstadt) | Beide Fragetypen wurden vollständig umgesetzt |
| Benutzeroberfläche | Intuitiv, kindgerecht, einfache Navigation | Klare Struktur mit Start-, Quiz- und Ergebnisfenster |
| Plattform | Windows-Desktop, offline-fähig, keine Installation notwendig | .exe-Datei funktioniert lokal und offline auf Windows |
| Technik | Python, Tkinter, JSON, Pillow | Alle Tools erfolgreich eingesetzt |
| Fragenanzahl | Mindestens 80 Fragen in strukturierter Form | Exakt 80 Fragen (40 je Fragetyp), gespeichert in JSON |
| Benutzerfreundlichkeit | Große Buttons, einfache Sprache, geeignet ab 6 Jahren | Erfolgreich getestet mit Kindern und Erwachsenen |
| Dokumentation | Entwickler- und Benutzerdokumentation geplant | Beide Dokumente in .txt und .pdf vollständig erstellt |
| Testkonzept | Eigener Testplan, strukturierte Durchführung | Umfassendes Testprotokoll dokumentiert (siehe Anhang) |

**Fazit:** Alle Anforderungen wurden erfolgreich umgesetzt oder sogar übertroffen.

Die Anwendung erfüllt sowohl funktionale als auch nicht-funktionale Kriterien vollständig.

## Aussichten

Obwohl das Projekt erfolgreich abgeschlossen wurde, bestehen mehrere Möglichkeiten für zukünftige Erweiterungen und Optimierungen:

1. **Online-Version (Web-App):**
   * Umsetzung mit HTML, CSS, JavaScript und einem Web-Framework (z. B. React, Flask)
   * Möglichkeit, Fragen dynamisch über eine Datenbank zu laden (z. B. MySQL, Firebase)
   * Plattformunabhängige Nutzung auf PCs, Tablets und Smartphones
2. **Mobile App (Android/iOS):**
   * Entwicklung mit Flutter, React Native oder Kotlin/Swift
   * Offline-Nutzung wie in der Desktop-Version
   * Möglichkeit, Sprachausgabe und kindgerechte Animationen einzubauen
3. **Mehrsprachigkeit:**
   * Erweiterung der JSON-Datenstruktur für zusätzliche Sprachen (z. B. Englisch, Arabisch, Französisch)
   * Umschaltbare Benutzeroberfläche für internationale Nutzer
4. **Benutzerkonto & Fortschrittsverfolgung:**
   * Speicherung des Fortschritts je Benutzer
   * Anzeige von Statistiken und Lernfortschritten
5. **Erweiterter Fragetypus:**
   * Neue Quiztypen (z. B. Länder → Kontinent, Hauptstadt → Flagge)
   * Zeitbasierte Herausforderungen oder Multiplayer-Modus

# Quellenverzeichnis

1. [Python Dokumentation – https://docs.python.org/3/](https://docs.python.org/3/)
2. [Tkinter GUI Referenz – https://tkdocs.com/](https://tkdocs.com/)
3. [Pillow Dokumentation – https://pillow.readthedocs.io/](https://pillow.readthedocs.io/)
4. [Tool zur Erstellung von .exe-Dateien](https://pyinstaller.org/) – pyinstaller.org
5. Eigene Erstellung von 80 Fragen basierend auf Wikipedia / EU-Websites
6. Flaggenbilder: frei verwendbar, aus Wikimedia Commons (<https://commons.wikimedia.org/>)

# Anhang

Im Anhang befinden sich alle ergänzenden Materialien und Dateien, die im Rahmen des Projekts „Länder-Quiz EU 1.0“ erstellt oder verwendet wurden.

|  |  |
| --- | --- |
| Wireframe Startfenster Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.  KI-generierte Inhalte können fehlerhaft sein. | Wireframe Benutzeranleitung Ein Bild, das Text, Screenshot, Display, Schrift enthält.  KI-generierte Inhalte können fehlerhaft sein. |

|  |  |
| --- | --- |
| Wireframe Quizfenster Ein Bild, das Text, Screenshot, Diagramm, Zahl enthält.  KI-generierte Inhalte können fehlerhaft sein. | Wireframe Ergebnisfenster Ein Bild, das Text, Screenshot, Rechteck, Schrift enthält.  KI-generierte Inhalte können fehlerhaft sein. |

## Relational\_Schema

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

## [country\_from\_flag – Land erkennen anhand der Flagge](#zurück_zu_Datenstruktur)

## Ein Bild, das Text, Screenshot, Schrift, Software enthält. KI-generierte Inhalte können fehlerhaft sein.

## [capital\_from\_country – Hauptstadt zum Land zuordnen](#zurück_zu_Datenstruktur)

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Screenshot der Startoberfläche

Ein Bild, das Text, Screenshot, Software, Betriebssystem enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Screenshot der Benutzeranleitung-Oberfläche

Ein Bild, das Text, Screenshot, Schrift, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

|  |  |
| --- | --- |
| Screenshot der Quiz-Oberfläche Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.  KI-generierte Inhalte können fehlerhaft sein. | Screenshot der Benutzeranleitung.txt Ein Bild, das Text, Screenshot, Schrift, Software enthält.  KI-generierte Inhalte können fehlerhaft sein. |

## Readme.md

Ein Bild, das Text, Screenshot, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Ein Bild, das Text, Screenshot, Schrift, Zahl enthält. KI-generierte Inhalte können fehlerhaft sein.Benutzweranleitung.txt

## Quill Code.py

Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Quill Code.py

Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Quill Code.py

Ein Bild, das Text, Screenshot, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Quill Code.py

Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Quill Code.py

Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Test Code.py

Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Ergebnis von Test Code.py

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Testprotokoll